

A High-Performance Communication Service for Parallel Computing on Distributed DSP Systems

Rapid increases in the complexity of algorithms for real-time signal processing applications have led to performance requirements exceeding the capabilities of conventional digital signal processor (DSP) architectures. Many applications, such as autonomous sonar arrays, are distributed in nature and amenable to parallel computing on embedded systems constructed from multiple DSPs networked together. However, to realize the full potential of such applications, a lightweight service for message-passing communication and parallel process coordination is needed that is able to provide high throughput and low latency while minimizing processor and memory utilization. This paper presents the design and analysis of such a service, based on the MPI specification, for unicast and collective communications.

Keywords – collective communications, digital signal processors, distributed systems, embedded systems, message passing

1. Introduction

With their emphasis on low power consumption and potent computational capability for signal processing applications, it is not surprising that DSPs have been employed in a multitude of applications. Similar to the general-purpose processor arena, the computational power of these special-purpose processors has continued to increase, providing the designer even more flexibility. However, many advanced signal processing applications continue to increase in complexity and require more computational power than a single processor can provide. To cope with these extreme demands, parallel processing techniques must often be employed.

Many applications targeted for DSPs, such as sonar array signal processing, are distributed in nature. Sonar system researchers have proposed using smart sensor nodes (i.e. each sensor node with its own processor) networked together in a distributed system to disperse the large computational burden imposed by the sensing algorithm [11-13]. Many of these remote-sensing applications require long distances between the sensing elements for accurate operation. Additionally, the sensing algorithms typically generate large amounts of inter-processor communication to distribute the computational burden among the smart nodes. Therefore, the network communication between these elements is just as critical to the performance of the application as the processing performed at the sensing locality. Although several systems have been proposed, the lack of proficient communication services for an embedded, distributed DSP system has limited the research to a general-purpose cluster of workstations.

Distributed computing has been extensively researched and proven a viable option for parallel applications. Several techniques have been explored to provide efficient communications between processing nodes, however, the standardization of the Message Passing Interface (MPI) specification [26] has placed it as the dominant choice for communication services required in distributed computing [19]. Consequently, MPI has been explored on nearly every network architecture available [5,9-10,15,18,23,30-31] and has the proven performance and functionality required for most parallel applications. Most of the research involved the use of general-purpose processors on a standardized network protocol. McMahon and Skjellum [25] did investigate the importance of reducing the full implementation for the limited memory space of an embedded system, however, their work did not take advantage of the hardware to provide the most efficient unicast and collective communications.

Analytical modeling of network performance has also been investigated extensively for distributed systems. Again, several techniques have been introduced, but the LogP [6] and LogGP [1] concepts have become a *de facto* standard. These models have been used to provide the basis for research in many areas of distributed computing, from assessment of network interface performance [7] to optimal broadcast and summation algorithms [21]. There are numerous examples available in a wide range of studies [8,22,24] that have used the LogP and LogGP models as a framework for performance and tradeoff analysis.

While widely examined for general-purpose systems, little research has investigated communications and synchronization services for special-purpose DSPs in arrays for distributed processing applications. This paper builds on proven techniques for general-purpose systems by providing a lightweight, MPI-compatible communication and coordination service for distributed DSP arrays assuming no network hardware support. The design leverages the architectural features of the DSP to provide low latency and high throughput on both unicast and collective communications. The system is compared to high-speed networks typically associated with distributed computing to evaluate its strengths and weaknesses. In addition, the LogP and LogGP framework is used to model the network communications to provide an accurate assessment of the design's performance and scalability. In doing so, the effects of improved processor clock rate and network bandwidth on several communication functions are assessed.

Section 2 describes the distributed DSP system architecture used as a basis for this study, while Section 3 describes the design of a communication service suitable for a wide range of DSP arrays. Section 4 compares the performance of the system and service against other network architectures and topologies generally employed in distributed computing systems. Next, the network performance is modeled and validated using the LogP and LogGP concepts as a framework in Section 5. Section 6 then explores varying model parameters for an enhanced system using the previous modeling techniques to examine the tradeoffs between clock rate and network throughput. Finally, Section 7 provides conclusions and suggested directions for future research.

2. Distributed DSP System Architecture

The similar application environments and design criteria for DSPs have caused many to converge to a reasonably common framework. They are not directly interchangeable, but the basic hardware primitives provided by DSP architectures for elementary communications (e.g. external access ports, integrated Direct Memory Access (DMA) controllers, and internal SRAM) are common to devices available from multiple vendors. The similarity of these features allows a lightweight communication service to be implemented on numerous DSP systems with minimal design changes.

For sensor arrays and other systems where it is desirable to disperse the processing and memory demands of the application across multiple nodes, a distributed architecture can be constructed by networking together multiple DSP nodes. The distributed architecture developed and employed in this research consists of multiple Bittware Blacktip-EX DSP development boards [4] connected to one another in a ring topology. Each board includes a single ADSP-21062 Super Harvard ARChitecture (SHARC) processor from Analog Devices as well as additional hardware for links to other nodes, off-chip memory, etc. In this work, the communication service was implemented and

optimized for this particular architecture, but implementation on other distributed architectures comprised of similar types of DSP processors, link ports, etc. would be relatively straightforward.

The SHARC processor, like most prevalent DSPs, integrates multiple link ports and their associated DMA controllers to provide high-speed, low-overhead communications. The bi-directional ports, each consisting of four data and two handshaking lines, can operate at twice the clock rate of the processor achieving a peak throughput of 40MB/s. Furthermore, the integrated DMA controllers require minimal setup and no processor overhead when communicating with other devices. Finally, the DMA channels assigned to each link port can operate concurrently, allowing both ports to transfer data simultaneously, thereby increasing the aggregate throughput of the system.

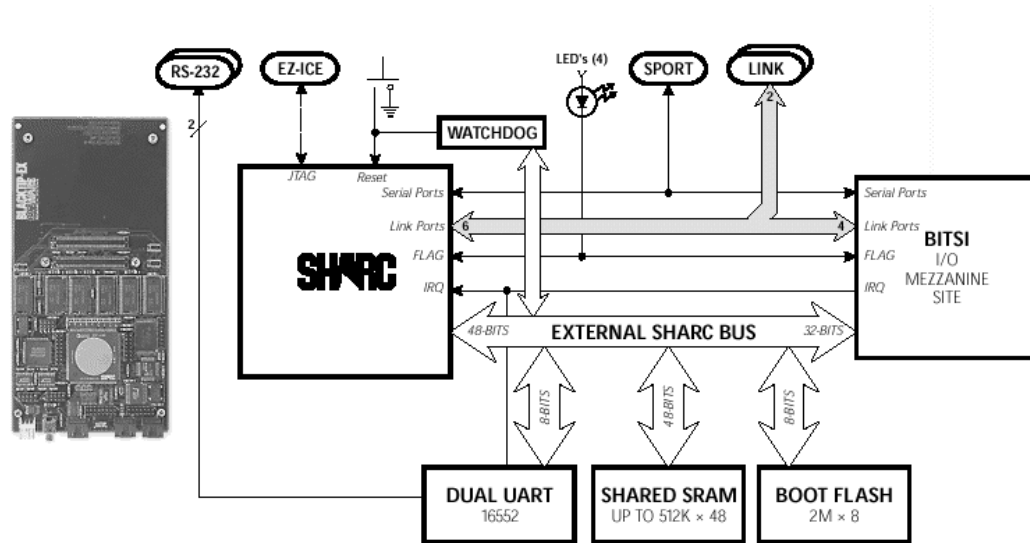


Fig. 1. Prototype Blacktip-EX development board and architecture [4].

Fig. 1 illustrates the Bittware Blacktip-EX development board and its architecture. In addition to 256KB of on-chip SRAM, the stand-alone prototype board includes 512K \times 48 bits of external SRAM especially useful for applications requiring large amounts of data memory. The board's non-volatile boot FLASH memory supports autonomous execution required for most embedded applications. The DSP can be accessed from any of the external interfaces including the BITSI mezzanine site, the serial ports, the link ports, the RS-232 interfaces, or the JTAG interface.

As seen in Fig. 1, the development board contains two link ports with external connectors to enable communications with other devices. To eliminate the need for external routing or switching hardware, the two link ports are dedicated to separate send and receive channels. This configuration allows the boards to be arranged in a uni-directional ring topology. While a ring does not provide the scalability of other topologies, its simple routing and low hardware complexity make it a natural choice for this system. In addition, if developed appropriately, certain message-passing functions can take advantage of a ring topology to produce highly efficient collective communications.

3. MPI Communication Service Design

MPI, like most other network-oriented middleware services, communicates data from one process to another across a network. TCP/IP sockets and other similar protocols deliver the same general functionality. However, MPI's higher level of abstraction provides an easy-to-use interface more appropriate for distributed, parallel computing applications. The MPI paradigm assumes a distributed-memory processing model, where each node has its own local address space and computes its data independently. The required data is explicitly exchanged between processing nodes using calls to library functions. The simplest and most common type of message exchange is the point-to-point or *unicast* communication. This communication involves exactly two processors in the system and requires matching send and receive functions at their respective nodes. Often, parallel processing applications require more complex communication distributions involving many or all nodes in a system. To meet this need, numerous collective functions are provided in the MPI specification. A classic example of a collective communication is the *broadcast*, where one node sends the same data to every other node in the system. The broadcast, as well as all of the collective functions, can be composed of multiple unicast (i.e. *multi-unicast*) transfers to provide the functionality specified. However, some systems can support efficient collective communication in hardware and therefore are able to exceed the performance of a multi-unicast transfer.

To provide the MPI functionality on an array of DSPs, the MPI-SHARC communication service was created. Since it was considered unfeasible to provide all 125 functions defined in the MPI specification on a simple embedded DSP, a reduced version of the specification was examined. Table 1 summarizes the most common functions used by parallel applications, and thus included in the design. Although MPI-SHARC is a subset of the full specification, the functionality and syntax is identical to MPI found on common distributed systems, allowing users to easily port applications developed on other platforms to an embedded, distributed DSP system.

Table 1
MPI functions included in MPI-SHARC

Function	MPI Name	Type	Purpose
Initialization	<i>MPI_Init</i>	Setup	Prepares system for message-passing functions.
Communication Rank	<i>MPI_Comm_rank</i>	Setup	Provides a unique node number for each processing element.
Communication Size	<i>MPI_Comm_size</i>	Setup	Provides number of nodes in system.
Finalize	<i>MPI_Finalize</i>	Setup	Disables communication services.
Send	<i>MPI_Send</i>	Unicast	Sends data to a matching receive.
Receive	<i>MPI_Recv</i>	Unicast	Receives data from a matching send.
Barrier Synchronization	<i>MPI_Barrier</i>	Collective	Used to synchronize all the nodes together.
Broadcast	<i>MPI_Bcast</i>	Collective	"Root" node sends same data to all other nodes.
Gather	<i>MPI_Gather</i>	Collective	Each node sends a separate block of data to the "root" node to provide an all-to-one scheme.
Scatter	<i>MPI_Scatter</i>	Collective	"Root" node sends a different set of data to all other nodes, providing a one-to-all scheme. Also known as a personalized broadcast.
All-to-All	<i>MPI_Allgather</i>	Collective	All nodes share their data with all other nodes in the system.

3.1. Unicast Communication

The design of the unicast communication, shown in Fig. 2a, is similar to protocols found in conventional networking schemes. A store-and-forward routing circulates the data and signaling packets around the ring. Although a more efficient wormhole routing technique might increase system performance, the inability of the link ports to provide a bypass function prohibits this approach. The storing of the data in intermediate nodes requires a dedicated memory buffer in the internal SRAM of the processor, thereby constricting the data payload size. To keep this memory requirement to a minimum, as well as increase performance, the variable payload size is restricted to a maximum of 2048B. If the message size exceeds this limit, multiple packets are sent in a packet-switching fashion, each with its own header packet providing a sequence number to ensure correct ordering. Finally, flow control is handled by a stop-and-wait protocol using a Ready-to-Send (RTS) packet from the receiver. The RTS signal ensures that the receiving node has allocated memory space before the sending node begins to transmit data.

Data packet headers and RTS packets share a common format, as shown in Fig. 2b. Both the source and destination fields provide 32 bits for addressing. However, addressing is assigned in a bit-wise fashion to allow multiple sources or multiple destinations to be designated by individual bits. This approach increases the flexibility of the header and RTS packet, but limits the system size to the number of bits in the field (i.e. 32 nodes in this case). To expand the design would simply require additional fields for the source and destination addressing.

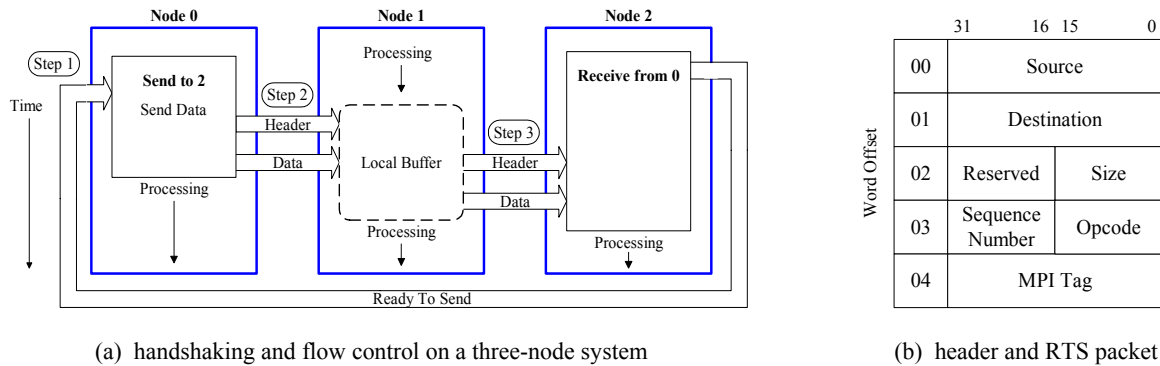


Fig. 2. Unicast communication in MPI-SHARC.

3.2. Collective Communication

Collective communication in MPI designs can be implemented in two ways. The simplest and most common is the organization of multi-unicast to provide the functionality of the collective transfer. Functions such as the *scatter* and *gather* require this implementation because of the nature of their underlying data distribution. Other functions, such as a broadcast or an *allgather*, can be implemented with multi-unicast as well, but performance benefits can be achieved on a system that provides a hardware-level broadcast through its network architecture. The MPI-SHARC communication service is designed to take advantage of the architecture whenever possible to provide efficient collective communication on the DSP array.

The design of the MPI-SHARC broadcast function, shown in Fig. 3, takes advantage of the collective property of a broadcast, the DSP hardware, and the ring topology. The collective nature of the MPI broadcast eliminates the header and RTS packets for data transmission. Both of these signaling packets are used to inform intermediate nodes, as well as the send and receive nodes, of the data traffic. Since a broadcast has a predetermined data distribution and network routing pattern, these signaling packets are superfluous and induce undesired overhead. However, to ensure all nodes are ready to receive and route the data, a barrier synchronization point is required internally at the beginning of the function. This barrier constitutes virtually all of the overhead in the broadcast. Efficiency and performance are increased by leveraging the use of multiple communication ports when a message size exceeds the 2048B limit. As shown in Fig. 3, a large message with multiple packets allows nodes to simultaneously send and receive data, thereby increasing the aggregate system bandwidth. Finally, the simple broadcast routing employed by a uni-directional ring eliminates the bottleneck most often seen at the root node with other topologies.

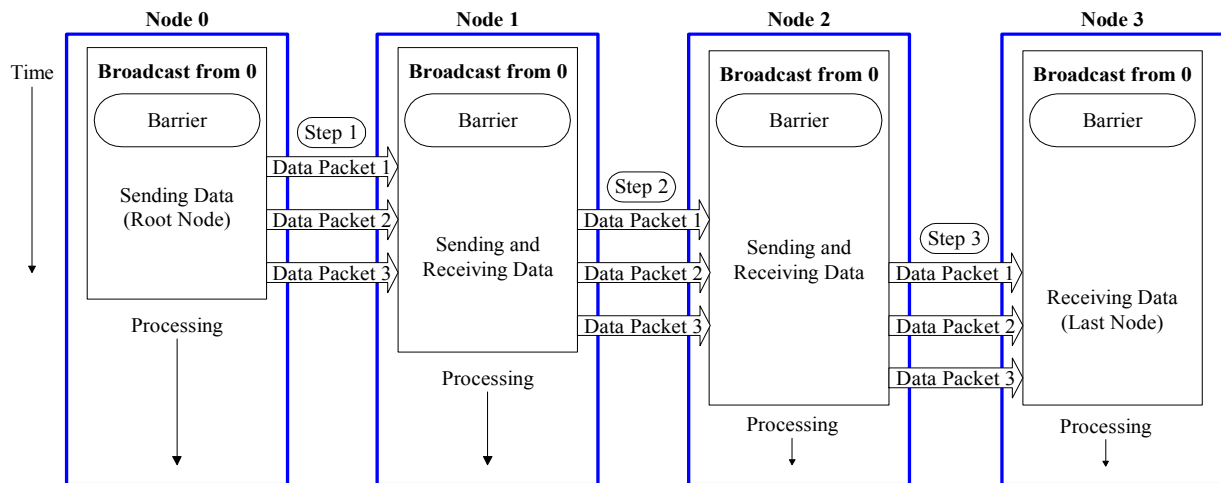


Fig. 3. Broadcast communication in MPI-SHARC.

Resembling the broadcast, the allgather in the MPI-SHARC design also exploits the DSP hardware, ring topology, and the function's inherent distribution pattern. Fig. 4 shows the data allocation that occurs during an allgather collective operation. Although this function could be made up of multiple broadcast invocations with modified data locations, it can also be implemented in a more effective fashion, facilitated by the separate send and receive ports as well as the ring topology. Fig. 5 demonstrates the operation of the allgather in the MPI-SHARC's design after the data has been relocated in the correct receive buffer. When performing the function, every node is sending and receiving data simultaneously, thereby increasing the system's aggregate throughput. Similar to the broadcast, the routing and distribution is predetermined and eliminates the header and RTS packet requirements. Additionally, a barrier synchronization point is used to assure that the nodes in the system are ready to perform the communication.

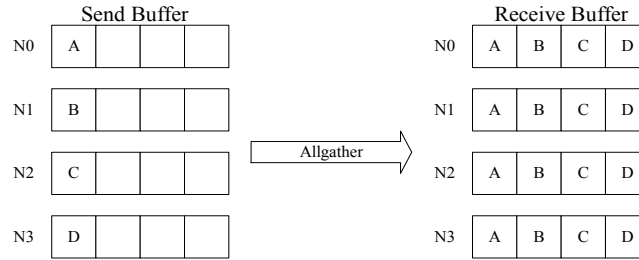


Fig. 4. Data movement with allgather communication [14].

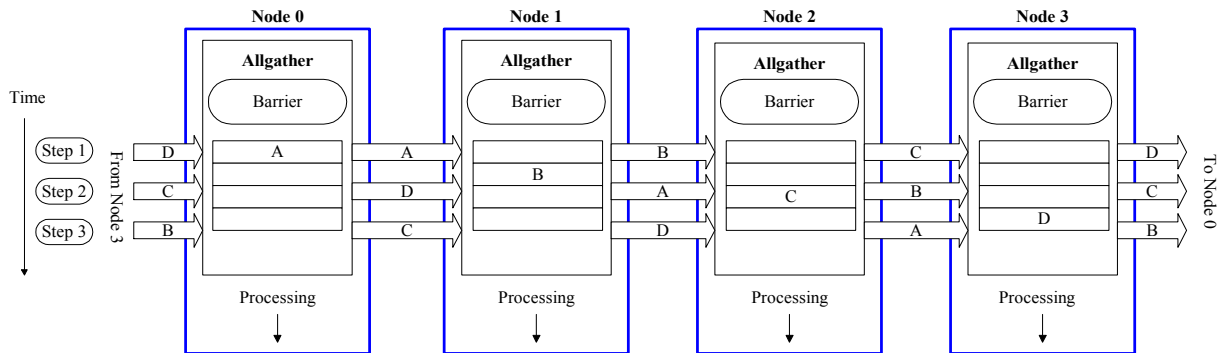


Fig. 5. Allgather communication in MPI-SHARC.

An additional technique employed to increase the performance of all-to-all communications is the implementation of an *in-place allgather*. This modified allgather, shown in terms of data allocation in Fig. 6, is relatively new and defined in the latest MPI specification [27]. While the actual amount of data transferred between nodes is identical in both cases, the performance and architectural improvements can be significant. A comparison of Fig. 4 and Fig. 6 shows the two distinct differences between the standard allgather and the in-place allgather. First, the reuse of the send and receive buffers is beneficial to a memory-restricted DSP, especially with large message sizes. Second, by assuring that the data to be transferred is already in the correct receive buffer location, the overhead induced moving the data from the send to the receive buffer is eliminated. In many cases, the code change required in the application using the MPI service is minimal and justified by the performance increase that is obtained.

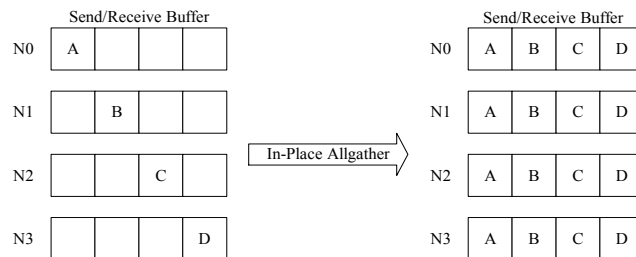


Fig. 6. Data movement with in-place allgather communication.

4. Performance Analysis

This section explores the performance of the MPI-SHARC design in comparison with several network architectures commonly associated with distributed, parallel systems. Included is a range of topologies, protocols, and MPI service implementations to provide a brief cross-sectional study of distributed network architectures. Additionally, every function included in the MPI-SHARC design is evaluated through direct testing or implied by an understanding of the underlying communication pattern. Investigation of these results demonstrates the network performance effects of the design issues introduced in the previous section.

4.1. Testbed Systems

Table 1 contains an overview of the cluster-based systems employed to compare and analyze the performance of the MPI-SHARC design. The network architectures listed can be grouped into two distinct categories. Clusters based on Fast and Gigabit Ethernet comprise the first class of systems in the list. Although these two similar network architectures provide services for distributed computing, their original design emphasis was on LAN functionality and interoperability for general-purpose computing applications. Consequently, the network stack and other architectural issues do not provide the low latency often required for a high-performance, distributed system. Systems in the second class are designed to directly address the issues needed to provide a low-latency, distributed, parallel architecture. Scalable Coherent Interface (SCI) and Myrinet both employ lightweight protocol stacks and low-level signaling techniques to achieve the low latency and high bandwidth required for high-performance, distributed computing.

Table 2
Testbed systems used for performance analysis

Network	Link Speed	Topology	Processor	Interface*	Memory	OS	MPI	Cost**
SHARC	320 Mbits/s	Ring	40 MHz SHARC DSP	N/A	256KB	N/A	MPI-SHARC 1.0	~\$1000 per node
Fast Ethernet	100 Mbits/s	Switched Star	400 MHz Pentium II	PCI 32/33	64MB	Linux 2.2.16	MPI Pro 1.5	~\$500 per node
Gigabit Ethernet	1000 Mbits/s	Switched Star	400 MHz Pentium II	PCI 32/33	64MB	Linux 2.2.16	MPI Pro 1.5	~\$1500 per node
SCI	3.2 Gbits/s	Ring	733 MHz Pentium III	PCI 32/33	128MB	Linux 2.2.17	SCALI ScaMPI 1.9.3.2	~\$2000 per node
Myrinet	1.28 Gbits/s	Switched Star	Dual 600 MHz Pentium III	PCI 64/66	256MB	Linux 2.2.17	Myricom GMPI 1.2.3	~\$2000 per node

The testbed systems were used to evaluate the unicast and collective MPI communication services expounded upon in the design section. The unicast, broadcast, and all-to-all functions were the focus in this study because of their dissimilar communication techniques which result in three unique performance trends. Conversely, the performance of the two remaining data transfer functions, the scatter and gather, can be deduced from the unicast

* The first number denotes the bus data width in bits, while the second number denotes the bus clock rate in MHz.

** These costs are estimates for all hardware during the timeframe when performance analyses were completed (Spring 2001).

results since they are composed using a multi-unicast arrangement. The differences between the unicast and collective operations also required separate bandwidth equations and timing measurement techniques.

Each experiment examined a full range of message sizes to assess trends in communication latency and bandwidth. Since the minimum message size is 4B (i.e. one 32-bit number), it was used for the low range of the scale. The restricted memory of the SHARC processor, and the stabilization of the resulting latency curves, established the upper message size at 16KB. To provide more concise results, only the minimum latencies and maximum bandwidths for each type of communication are shown. Typically, the minimum latency occurred at the 4B message size, while the maximum bandwidth was measured at the 16KB message size. However, this situation was not always the case because of various network design and data distribution issues associated with the cluster systems. The exact cause of these exceptions is beyond the scope of this research, but their occurrence is noted in the results.

The following sections explore the latency and bandwidth results on common four- and eight-node system sizes. To represent the range of operating characteristics, the results include unicast and collective communication measurements, as well as measurements taken from an application case study in sonar signal processing.

4.2. Unicast Experiments and Results

To achieve accurate measurements, unicast latency and bandwidth were determined using a ping-pong test. Evaluation of a single send and receive pair will produce contrasting communication times depending on their location in the ring. For example, a unicast transfer to an adjacent node will produce a shorter latency than one that makes several hops around the ring to reach its destination. The unilateral transversal of the entire ring ensured by the ping-pong test provides an average latency linearly proportional to the system size resulting in larger latencies and lower bandwidths as the system size increases. This size dependence is an obvious downfall of a ring topology that affects its unicast scalability. By contrast, network distance in a switched network (i.e. Ethernet and Myrinet in this case) is constant, resulting in latencies and bandwidths that are independent of the number of nodes.

Table 3
Unicast communication results

Network	Minimum Latency (μ secs)	Maximum Bandwidth (MB/s)
Four-Node SHARC	81.38	19.05
Eight-Node SHARC	129.98	16.95
Fast Ethernet	183.40	9.65
Gigabit Ethernet	157.11	18.15
Four-Node SCI	5.14	64.69
Eight-Node SCI	5.32	63.87
Myrinet	15.74	81.21

The MPI-SHARC results, shown in Table 3, exhibit performance comparable to, or exceeding, both Ethernet-based systems. The four-node SHARC system exhibits a latency that is 51.8% of Gigabit Ethernet, while an eight-node system is still less, but increases to 82.7%. The maximum bandwidth is also comparable to Gigabit Ethernet

but, like the latency, it is negatively affected by the system size. This trend matches the intuitive expectations of an increased hop count associated with a larger ring size causing an increase in latency and decrease in bandwidth. SCI, which also is arranged in a uni-directional ring topology, exhibits similar behavior, although not as drastic. The minimum latencies and maximum bandwidths of the Ethernet systems do not display this phenomenon, and eventually the star architecture's performance will surpass the ring topology of the MPI-SHARC in both metrics. By contrast, the four- and eight-node MPI-SHARC design cannot compete with the high performance SCI and Myrinet systems in terms of latency or bandwidth for unicast communication on the system sizes tested.

4.3. Broadcast Experiments and Results

To evaluate broadcast performance, results are measured at the node that incurs the maximum latency. In many distributed applications, the computational requirements are equally distributed and therefore the overall system performance will ultimately be determined by the distribution of the data to the last node. The example in Fig. 7 demonstrates the unequal communication latencies reducing the computation time available at each processor. Node 1, shown in the example, will complete its data processing before any other node and will be required to wait idle for the other nodes to complete. Consequently, the node that receives its data last will complete its processing last, and thus determine the overall processing time of the parallel system.

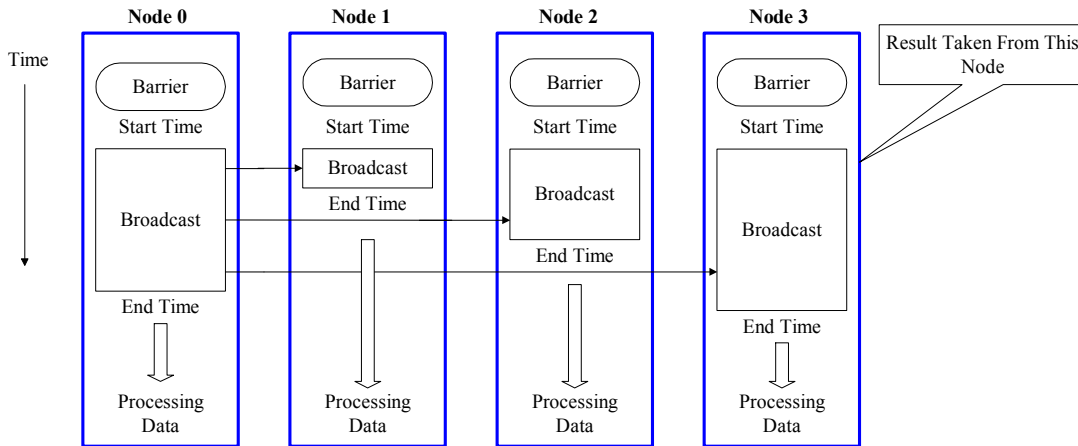


Fig. 7. Example of uneven broadcast latency and result measurement point.

While latency results can be directly assessed, bandwidth of the parallel system is calculated from the message size and latency measurements as follows:

$$Bandwidth = \frac{Message\ Size \times (P - 1)}{Latency} \quad (1)$$

The bandwidth shown in this equation incorporates the total amount of data transferred as well as the latency results. Since the broadcast is a collective function that distributes data to every node in the system, it is dependent on P , the number of *processors* or *nodes*.

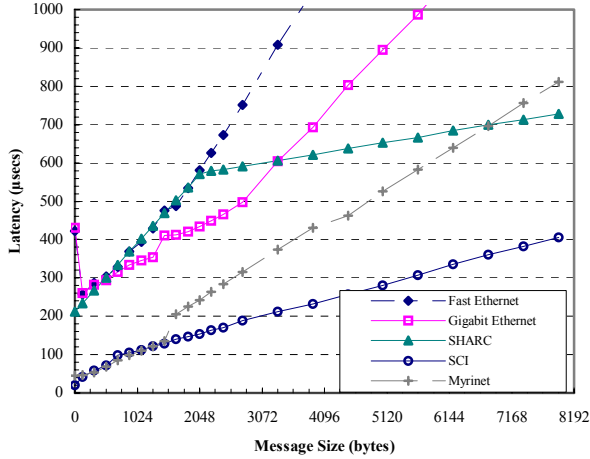
As seen in Table 4, the latency of MPI-SHARC is only 64% and 83% that of Fast and Gigabit Ethernet for a four- and eight-node system, respectively. While relegated to a much slower host processor, MPI-SHARC avoids the overhead of processing a heavyweight protocol stack and thus latency is somewhat reduced. However, it still cannot compete with the network architectures in the high-performance category, achieving at least 8.5 times the latency of SCI and 3.7 times that of Myrinet, since these systems feature both lightweight protocols and fast processors. However, the bandwidth has a much different outlook. Since the MPI-SHARC design implements a form of true simultaneous broadcast, the achieved bandwidth is comparable to the high-performance network architectures and far surpasses both types of Ethernet. For instance, the results show a bandwidth performance almost four times that of Gigabit Ethernet in an eight-node system.

Table 4
Broadcast communication results

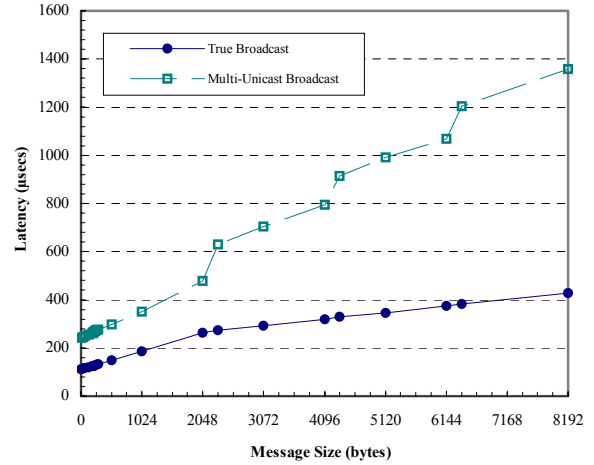
Network	Four-Node System		Eight-Node System	
	Minimum Latency (μsecs)	Maximum Bandwidth (MB/s)	Minimum Latency (μsecs)	Maximum Bandwidth (MB/s)
SHARC	110.35	75.99	212.43	150.06
Fast Ethernet	173.48	17.43	257.48	27.10
Gigabit Ethernet	173.21	33.69	260.61	40.56
SCI	13.05	92.55	20.77	150.06
Myrinet	29.97	137.62	44.54	122.89

To further investigate the intricacies of the MPI-SHARC broadcast design, the full results for the larger system size are plotted in Fig. 8a. While most of the systems maintain a fairly linear behavior, inspection of the SHARC results shows a distinctive two-piece linear trend. This characteristic is caused by the message size exceeding the maximum payload of 2KB and the send and receive ports operating simultaneously in a packet switching fashion. Since the slopes of the latency curves in Fig. 8a are directly proportional to system bandwidth, the decrease of 558% seen on the eight-node SHARC system corresponds to an overall increase in system bandwidth.

The support of a hardware broadcast with this design allows investigation into the performance improvements over the common multi-unicast scheme. The two distribution patterns were tested on a four-node SHARC system and the results plotted in Fig. 8b. The initial latency of the hardware broadcast is 45% that of the multi-unicast and the curves diverge quickly, especially when the message size exceeds the 2KB maximum payload size. Once the message size has reached the 8KB limit shown in Fig 8b, hardware broadcast is over three times faster. The noticeably different trends of the slopes will cause this value to increase further at larger message sizes. Not only is the latency greatly decreased, but since system bandwidth is directly related to the slope of the latency curve, a substantially higher bandwidth is achieved using a system that implements a hardware broadcast scheme over a multi-unicast arrangement.



(a) broadcast results on the eight-node systems



(b) true broadcast vs. multi-unicast on a four-node SHARC system

Fig. 8. Expanded broadcast results.

4.4. All-to-All Experiments and Results

Although the underlying distribution techniques are different, many similarities associated with collective communications can be seen between the all-to-all and broadcast functions. For applications where computation is equally distributed across processors, all-to-all results can be measured at the node incurring the maximum latency. These results are then used to assess the overall system performance. The bandwidth is calculated as follows:

$$Bandwidth = \frac{Message\ Size \times (P - 1) \times P}{Latency} \quad (2)$$

Table 5

All-to-all communication results

Network	Four-Node System		Eight-Node System	
	Minimum Latency (µsecs)	Maximum Bandwidth (MB/s)	Minimum Latency (µsecs)	Maximum Bandwidth (MB/s)
SHARC	147.75	48.74	260.33	157.25
SHARC In-place	147.28	143.39	258.78	288.09
Fast Ethernet	1096.27	11.42	2079.03	19.65
Gigabit Ethernet	961.48	18.53	1867.15	28.53
SCI	30.96	152.60	71.44	284.07
Myrinet	83.35	140.53	670.88	71.51

As computed from measurements using Eq. 2 and shown in Table 5, the performance achieved with the MPI-SHARC design is now comparable and, in some cases, transcends the class of high-performance architectures. The minimum latency of the SHARC measures 15.3% and 13.8% of the four- and eight-node Gigabit Ethernet testbeds, respectively. Although in most cases the latency is not on par with the high-performance SCI or Myrinet architectures, it is much closer to these systems than their Ethernet counterparts. Similar to the broadcast, the

bandwidth measurements from the MPI-SHARC design exhibit results comparable to the high-performance category and surpass those of the Ethernet systems. For instance, the MPI-SHARC bandwidth is 7.7 and 10.1 times that of Gigabit Ethernet for a four- and eight-node system, respectively.

In both system sizes, the maximum bandwidth of the in-place allgather significantly outperforms the standard allgather. As shown earlier, the allgather must first move data internally from the send buffer to the appropriate location in the receive buffer. While DSPs excel at signal processing, they typically perform rather poorly using a standard compiler's implementation of a memory move. This additional overhead is rather substantial, greatly affecting performance at large message sizes.

4.5. Parallel Application Experiments and Results

Although the raw latency and bandwidth measurements have been compared, the final target for this communication service is for use by applications. Several computational-intensive beamforming algorithms have been proposed for this type of embedded, distributed DSP system. The Matched-Field Tracking (MFT) algorithm [20] is a typical example of an application that would benefit from a configuration made possible by the MPI-SHARC design. The MFT algorithm processes passive sonar data from an array of hydrophones to produce a three-axis tracking result of an unknown moving target in an undersea environment. It has been implemented and analyzed on the SHARC array as well as two of the Ethernet-based clusters, which allows a comparison to the MPI-SHARC design. The code from the parallel algorithm employed here requires approximately 20,000 instances of a 40B allgather communication during its execution.

It is difficult to directly compare the execution times between testbed systems because of the vast differences in processor speeds. Instead, Eq. 3 defines efficiency of a parallel system, where *Ideal Speedup* is equal to the number of processing elements used for computation. Eq. 4 defines the *Measured Speedup* used in Eq. 3. Here, parallel efficiency normalizes the effects of processor speed and thereby focuses on the communication and parallel decomposition techniques. In general, a higher efficiency equates to less overhead in the MPI communication, which allows designers to implement more flexible parallel algorithms.

$$\text{Parallel Efficiency} = \frac{\text{Measured Speedup}}{\text{Ideal Speedup}} \quad (3)$$

$$\text{Measured Speedup} = \frac{\text{Sequential Execution Time}}{\text{Parallel Execution Time}} \quad (4)$$

Fig. 9 shows the results of the parallel MFT algorithm implemented and executed on three testbed systems. The MPI-SHARC design exhibits superior performance and scalability compared to both of the Ethernet systems. For instance, for four nodes, the SHARC system achieves twice the efficiency of the Ethernet systems, and as the size of the system increases, the efficiency of the SHARC system drops much less rapidly.

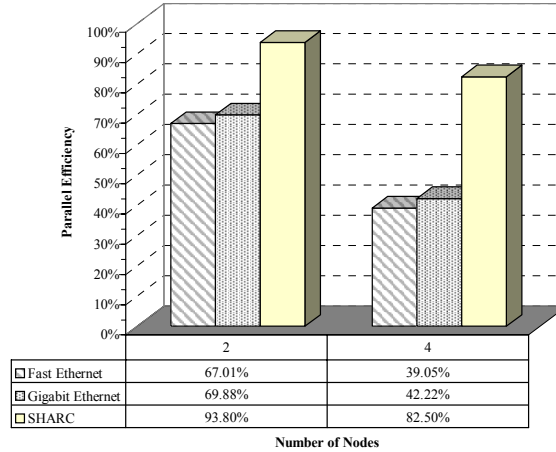


Fig. 9. Parallel efficiency of parallel MFT algorithm.

5. System Modeling and Validation

To explore the effects of system parameter tradeoffs on the MPI-SHARC communication service without implementing the costly physical hardware requires the use of a model. While many forms of interconnection performance models have been proposed, the LogGP model, which is a superset of the LogP model, most accurately conforms to the characteristics of the MPI-SHARC communication service and was adopted and extended. LogGP models have been employed in the literature on a wide range of algorithms and architectures and found reliable for many situations and applications and their flexibility is well suited to MPI-SHARC service's design and performance characteristics. This pliancy, which allows the designer to reduce or expand the model's parameters if the system approximations are justifiable [1,6-7], is used at length when modeling the service's functions.

The LogP framework attempts to model a distributed system's communication through the use of a minimal number of parameters. LogGP expands upon the parameters of LogP in an attempt to incorporate support for systems with increased bandwidth for large messages. Additionally, extensions to the original LogGP model parameters are required to accurately model particular functions of the MPI-SHARC communication service. A complete list of the parameters is defined in Table 6.

The original goal of the LogGP model was to provide a performance estimate of parallel network communications on real machines using a reasonable set of parameters. The designers of the model considered it a compromise to provide sufficiently accurate results using a minimum number of parameters that could easily be determined on a wide range of current and future parallel machines. As a result, several assumptions were made in the original LogGP framework that conflict with the behavior of the MPI-SHARC design. To account for these discrepancies, parameters are either simplified or augmented to accurately model each function's characteristics.

Table 6
LogGP model parameter definitions for the MPI-SHARC communication service

Parameter	Symbol	Definition
<i>Send Latency</i>	L_s	The length of time incurred in the network routing the header and data from the sending to the receiving node; during this time the processor can perform other operations.
<i>RTS Latency</i>	L_{rts}	The length of time incurred in the network routing the RTS packet from the receiving to the sending node; during this time the processor can perform other operations.
<i>Send Overhead</i>	o_s	The length of time the sending processor is engaged in the transmission of data; during this time the processor cannot perform other operations.
<i>Receive Overhead</i>	o_r	The length of time the receiving processor is engaged in the reception of data; during this time the processor cannot perform other operations.
<i>Additional Packet Overhead</i>	o_{ap}	The length of time induced by each additional packet at the receiving and sending processors; during this time the processor cannot perform other operations.
<i>gap</i>	g	The minimum time interval between consecutive small message transmissions or consecutive small message receptions at a processor. The reciprocal of g corresponds to the available per processor communication bandwidth for small messages.
<i>Gap per byte</i>	G	The minimum time interval between consecutive large message transmissions or consecutive large message receptions at a processor. The reciprocal of G corresponds to the available per processor communication bandwidth for large messages.
<i>Send Hops</i>	h_s	The number of hops from the sending to the receiving node; equivalent to the number of hops experienced by the header and data packets.
<i>RTS Hops</i>	h_{rts}	The number of hops from the receiving to the sending node; equivalent to the number of hops experienced by the RTS packet.
<i>Number of Processors</i>	P	The number of processors in the system.
<i>Minimum, Maximum Packet Payload Size</i>	w_{min}, w_{max}	The minimum and maximum packet payload size, in bytes, respectively.
<i>Message Size</i>	k	The size of the message, in bytes, being transmitted.

5.1. Unicast Communication Model

The modeling of the unicast function can be divided into two possible scenarios. The first is the case when the message size is less than the 2KB limit, thereby requiring only one packet. When the message exceeds the 2KB limit, a more complex model is necessary to account for the multiple packets transmitted. However, to simplify the second, the first model is used as a baseline and expanded upon to provide the required operating characteristics. Both of these possible cases require more parameters than originally defined in the LogGP framework to provide accurate results.

The single-packet unicast model is built on the transmission of a minimum-sized message as shown in Fig. 10. In this particular example, data is sent from node 4 to node 1. The transmission of a packet requires overhead at the sending and receiving nodes to formulate and decode the packet, as well as the latency of the data packet and RTS signal through the network. Assuming both the sending and receiving nodes begin their respective functions simultaneously, the time required for a minimum-sized packet at the receiving node will equal $L_{rts} + o_s + L_s + o_r$. As a result of the communication service's design, the *RTS Latency*, *send overhead*, and *receive overhead* are

independent of packet payload size. The *Send Latency*, L_s , however, is only defined for latency of a minimum-sized payload. If the message is larger than the minimum, the time per byte, or *gap*, must be added for the subsequent bytes exceeding the minimum payload size. Thus, the model for the time required for a message size less than the maximum packet payload size, w_{max} (i.e. requiring only one packet) is shown in Eq. 5. Additionally, the location of the sending and receiving nodes must be addressed by the model to accurately characterize the *RTS Latency* and *Send Latency* parameters. To account for the location of the nodes, two additional hop-count parameters, defined in Table 6, are integrated into the L_{rts} and L_s terms in the model parameters table.

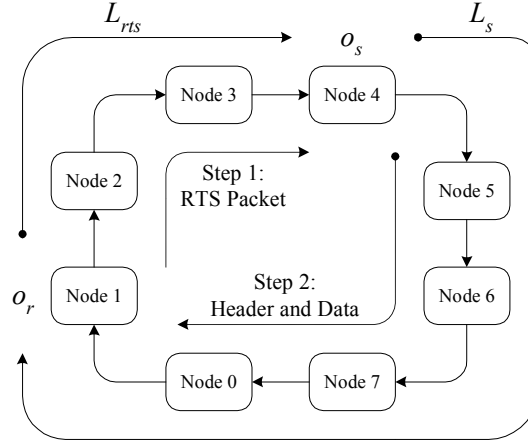


Fig. 10. Example of a unicast transfer from node 4 to node 1.

When a message exceeds the 2KB maximum, thereby dictating the use of multiple packets, a more complex model is required. Since packet switching is used, the communication time is based on an initial maximum-sized packet payload, $L_{rts} + o_s + L_s + o_r + (w_{max} - w_{min}) \times g$, followed by overlapping additional packets. This characteristic requires the overhead of processing each additional packet, o_{ap} , defined in Table 6, and an increased bandwidth term, *Gap per byte*, resulting from the overlapping transmission of packets. The floor function is used to determine the number of additional packets after the first initial maximum-sized packet and multiplied by the additional overhead required. The *maximum packet payload size*, w_{max} , is then subtracted from the total *message size*, k , since the increased bandwidth term, G , is only valid for the data bytes after the initial maximum-sized packet. Incorporating these terms yields Eq. 6 for calculating the communication time requirement for the transfer of messages that exceed the 2KB limit. Finally, by including minimum and maximum functions, Eq. 6 can be made to reduce to Eq. 5 resulting in a single model for any size message, as shown in Eq. 7.

$$T_{unicast_small} = L_{rts} + o_s + L_s + o_r + (k - w_{min}) \times g \quad (5)$$

$$T_{unicast_large} = L_{rts} + o_s + L_s + o_r + (w_{max} - w_{min}) \times g + \left\lfloor \frac{k}{w_{max}} \right\rfloor \times o_{ap} + (k - w_{max}) \times G \quad (6)$$

Or, in general

$$T_{unicast} = L_{rts} + o_s + L_s + o_r + \min \{ (k - w_{min}), (w_{max} - w_{min}) \} \times g + \left\lfloor \frac{k}{w_{max}} \right\rfloor \times o_{ap} + \max \{ 0, (k - w_{max}) \} \times G \quad (7)$$

Systems sizes of two, three, and four nodes were used with various measurement techniques to determine an average value for each model parameter. While most of the parameters listed in Table 6 appear directly in the equations above, several are more clearly presented as part of the measured model parameters in Table 7.

Table 7
Measured model parameters for unicast communication

o_r	o_s	o_{ap}	L_s	L_{rts}	g	G	w_{min}	w_{max}
29.17 μsecs	28.63 μsecs	48.30 μsecs	$h_s \times 15.40$ μsecs	$h_{rts} \times 8.925$ μsecs	$(h_s + 1) \times 0.025$ μsecs	0.025 μsecs	4B	2048B

5.2. Broadcast Communication Model

The broadcast scheme used in the MPI-SHARC design is a true collective function and not formally defined in a LogGP model. As a result, the derivation of a new model uses the concepts introduced in the LogGP framework but is based more upon the broadcast's operating characteristics and an examination of the measured results. As shown earlier, the function produces a two-piece linear curve, thereby simplifying the complexity of the model and the required number of parameters.

Although there is a resemblance between the broadcast and unicast functions because of the packet switching employed, other underlying operating factors simplify the broadcast model. Since the broadcast does not implement additional header packets with each data packet or require an RTS signal, the model eliminates the need for the o_{ap} and L_{rts} parameters. Since the overhead induced at each node is an effect of a barrier synchronization between nodes, it is relatively constant and does not warrant separate *send* and *receive overheads*. These reductions produce a single *overhead* term, o , that is identical at every node in the system. The broadcast operates much like a unicast communication that must transverse the entire ring, and as a result, the *gap* and *Latency* terms included in the models shown in Eqs. 8, 9, and 10 exhibit similarities to the unicast models. The measured parameters, given in Table 8, also closely resemble the values for the unicast. However, because no header packet is used for each data packet, the L_s term is greatly reduced.

$$T_{broadcast_small} = o + L_s + (k - w_{min}) \times g \quad (8)$$

$$T_{broadcast_large} = o + L_s + (w_{max} - w_{min}) \times g + (k - w_{max}) \times G \quad (9)$$

Or, in general

$$T_{broadcast} = o + L_s + \min \{ (k - w_{min}), (w_{max} - w_{min}) \} \times g + \max \{ 0, (k - w_{max}) \} \times G \quad (10)$$

Table 8
Measured model parameters for broadcast communication

o	L_s	g	G	w_{min}	w_{max}
$63.25 + (P-2) \times 23.10 \mu\text{secs}$	$(P-2) \times 1.6 \mu\text{secs}$	$(P-1) \times 0.025 \mu\text{secs}$	$0.0266 \mu\text{secs}$	4B	2048B

5.3. All-to-All Communication Model

The underlying operation of the two all-to-all functions provided in the MPI-SHARC design further simplifies the model required to obtain accurate results. In the all-to-all functions, every node is simultaneously sending data to the adjacent node in a predetermined distribution pattern. As a result, there is no advantage to splitting large messages into multiple smaller packets. The concurrent, balanced data transmission at each node also ensures that each node completes the function simultaneously. Consequently, each node produces an identical one-piece linear curve with an initial *overhead* offset.

The only difference between the two all-to-all functions, allgather and in-place allgather, is the data move required by the allgather that incurs an equal overhead at every node which is linearly related to the message size. Thus, one extra *overhead* parameter, *memory move overhead*, o_{mm} , is added to the all-to-all model, but reduces to zero with the in-place allgather. Eq. 11 displays the resulting model and Table 9 lists the measured parameters.

$$T_{alltoall} = o + k \times (g + o_{mm}) \quad (11)$$

Table 9
Measured model parameters for all-to-all communication

Function	o	o_{mm}	g
Allgather	$91.10 + (P-2) \times 27.50 \mu\text{secs}$	$0.1627 \mu\text{secs}$	$(P-1) \times 0.025 \mu\text{secs}$
In-place allgather	$91.10 + (P-2) \times 27.50 \mu\text{secs}$	$0 \mu\text{secs}$	$(P-1) \times 0.025 \mu\text{secs}$

One interesting item to note in the model parameters is the relatively large value measured for *memory move overhead*, o_{mm} . This additional overhead is 6.5 times larger than the network communication time of a two-node allgather, demonstrating the phenomenon of the overhead of the processor's internal memory move incurring more time than the transfer of the same amount of data over the network. As shown previously in the results, this characteristic is the cause of a major performance handicap when using the allgather function.

5.4. Model Validations

Fig. 11 displays the validation with respect to message and system size for the unicast model. While the message size comparison exhibits a slight divergence around 5120 bytes, the error never exceeds 2.25%. The system size validation displays the same high degree of precision. With small messages, the largest deviation occurs with a two-node system measuring a 1.42% error, while with large messages the maximum variation of 3.00% error is seen on a three-node system.

Although not shown, the collective broadcast and all-to-all models exhibit promising results with the same high degree of accuracy. The broadcast model produces a maximum of 2.52% error when validated with message size and a maximum of 2.90% error when validated with system size. The all-to-all model experiences maximum errors

of 2.62% and 2.68% for the message size and system size validations, respectively. For both models, the operating characteristics are similar to the unicast results in Fig. 11 and do not exhibit a divergence with increasing message or system size.

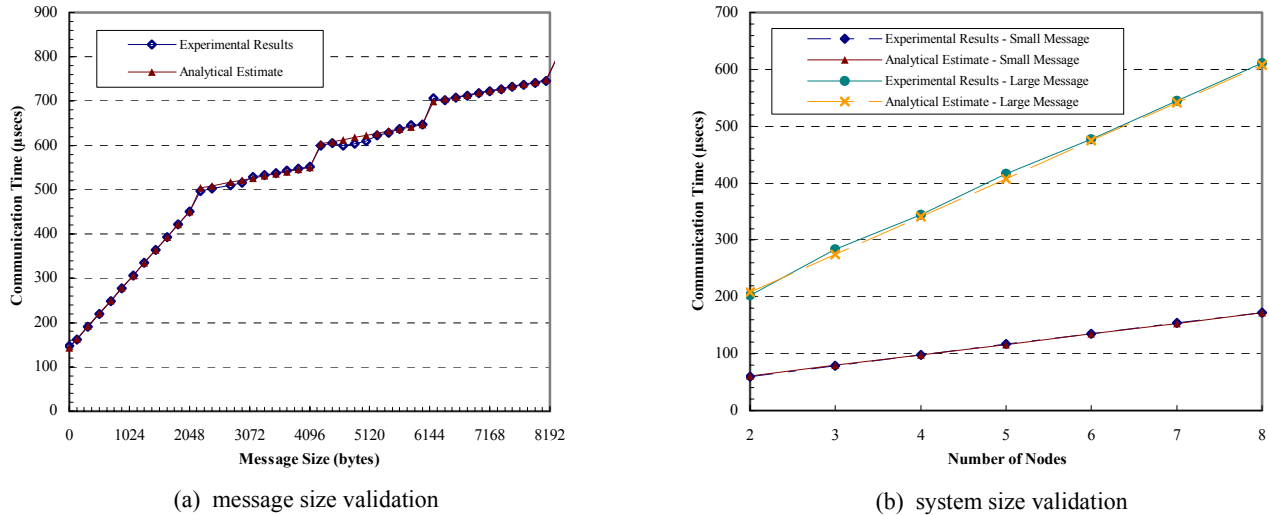


Fig. 11. MPI-SHARC unicast model validation.

6. System Tradeoff Analysis

With a model for the MPI-SHARC communication service, it is possible to ascertain performance with various tradeoffs in the system architecture. Consistent with the trend in general-purpose processors, the SHARC and other DSP processor families are constantly being improved and superceded by faster designs. The next generation of the SHARC, the TigerSHARC, increases the processor clock rate to 150 MHz and the link port bandwidth to 150 MB/s. Although the clock rate increases by almost a factor of four, the basic underlying processor architecture and interface hardware remain consistent providing backward capability with current designs.

While the updated processor can operate at higher clock and data throughput rates, several factors may determine the optimal values for a deployable system. While an increased clock rate will reduce the processing time of the algorithm, it will also harshly affect the power consumption. The clock rate versus power dissipation tradeoff is typically a significant design issue that needs to be examined for a battery-powered system such as an autonomous sonar array. Additionally, the maximum speed of the link ports, 1.2 Gbits/sec, may pose significant feasibility issues in a large, distributed system. The physical distance between nodes and the attachment of slower devices to the network will limit the maximum throughput obtainable in a deployable system. An algorithm's communication requirements determines whether this tradeoff causes significant degradation of overall parallel performance or only a minor, acceptable decline.

Microbenchmarking an application's requirements on a single processor will allow system designers to accurately determine the computational performance tradeoffs of varying the processor clock rate. However, the complexity of the communication service requires the use of a model that factors in more parameters to examine the

tradeoffs of both the processor speed and the network throughput. Several design features of an embedded processor allows assumptions to be made that simplify this analysis. The most notable is the use of dedicated, internal SRAM allowing the processor to fetch and execute all instructions and data at the same rate as the processor clock. This design is in major contrast to most general-purpose processors which become limited by a slower external memory bus and DRAM architectures. Secondly, there is no secondary storage or complicated video requirements to limit the system's performance when the clock rate of the processor is increased. As a result, the assumption is made that an increase in processor clock rate is linearly proportional to a system's computational performance.

Therefore, based upon the previous assumption, an increase in clock rate will linearly decrease all of the overhead parameters in the MPI-SHARC communication service. These overheads are a direct result of DMA controller setup, packet formulation and decoding, data moves, etc. required for the communication service and are directly dependent on processor execution speed. Additionally, the latency terms in the point-to-point model will be linearly affected by the clock rate because of its dependence on the processing time required to store and forward the data through a node. It can be argued that the bandwidth of the link ports also affects the *Send Latency*. While it does contribute a small factor, with the minimum 4B message the data rate accounts for only 0.65% of the total *Latency* time.

As stated earlier, the maximum peak bandwidth of the SHARC's link ports is 40 MB/s, or 0.025 $\mu\text{sec}/\text{byte}$. The *gap* and *Gap per byte* parameters listed in the previous section consistently measure 0.025 $\mu\text{sec}/\text{byte}$ or an integer multiple of 0.025 $\mu\text{sec}/\text{byte}$. Therefore, the second justifiable assumption is that an increase in the data rate of the link ports will linearly decrease the *gap* and *Gap per byte* model parameters.

With these two assumptions, the LogGP model parameters are placed into two distinct categories depending on whether they are affected by data rate or clock rate. The model parameters are then updated to reflect these two new criteria, B the *link port bandwidth* measured in MB/s, and f_{clk} the *clock rate* measured in MHz. The model remains identical, however the parameters have been updated with these new terms. For example, the updated parameters used for the system enhancement comparisons are shown in Table 10.

Table 10
Updated parameters for the point-to-point model to include *clock rate* and *link port bandwidth*

o_r	o_s	o_{ap}	L_s	L_{rts}	g	G	w_{min}	w_{max}
$\frac{1166.8}{f_{clk}}$ μsecs	$\frac{1145.2}{f_{clk}}$ μsecs	$\frac{1932}{f_{clk}}$ μsecs	$h_s \times \left(\frac{616}{f_{clk}}\right)$ μsecs	$h_{rts} \times \left(\frac{357}{f_{clk}}\right)$ μsecs	$(h_s + 1) \times \left(\frac{1}{B}\right)$ μsecs	$\frac{1}{B}$ μsecs	4B	2048B

Using the updated model parameters, four different configurations were investigated. The SHARC and TigerSHARC systems both use their maximum clock rate and network data rate, at 40 MHz and 40 MB/s for the SHARC and 150 MHz and 150 MB/s for the TigerSHARC. The next two systems investigated variations of the two maximum values for processor clock rate and network data rate. The *enhanced clock rate system* increased the processor clock to 150 MHz, while maintaining the 40 MB/s data rate. Conversely, the *enhanced data rate system* increased the network bandwidth to 150 MB/s, while keeping the processor clock rate at 40 MHz.

All four systems are first evaluated with respect to message size on an eight-node system. While the scales used in Fig. 12 vary depending upon the communication type, the same range used in the original performance analysis of Section 4 (i.e. 4B to 16KB) was investigated to determine the area of relevance.

As seen in Fig. 12, increasing the clock rate of the processor reduces the initial overhead time in all types of communications. However, as the message size increases, the data rate begins to dominate the communication time and a crossover point is observed. In the broadcast, a significant divergence above the crossover point is observed in Fig. 12b between the enhanced clock rate and enhanced data rate system. However, in the unicast results of Fig. 12a, their interpolated linear slopes are relatively equal with a slight offset. This behavior is a result of the additional overhead required for each additional packet in the unicast communication. The enhanced clock rate system decreases this overhead term but takes longer to transfer the data, while the enhanced data rate speeds up the data transmission but takes longer to process each packet. These contrasting effects result in comparable performance trends.

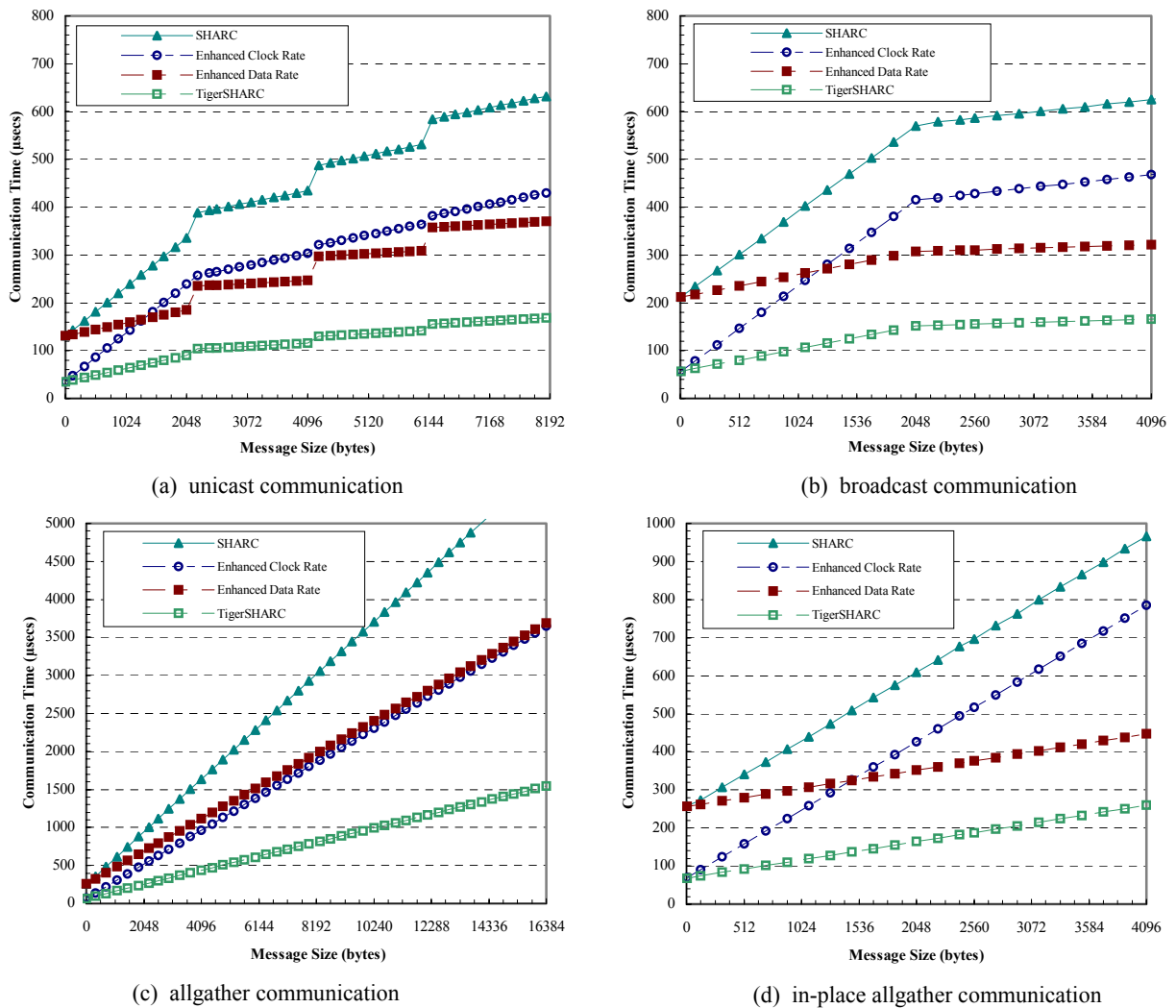


Fig. 12. Message size analysis of unicast, broadcast, and all-to-all communication.

Fig. 12 shows two dissimilar situations for the all-to-all communication functions. The in-place allgather results in Fig. 12d resemble the broadcast results, where an increase in clock rate reduces the initial overhead but eventually the data rate becomes more dominate and a crossover point is observed. While a crossover does occur with the allgather, the message size that it occurs at is 11 times higher and, unlike the other functions, an increase in clock rate is always more effective up to a 16KB message size. This phenomenon is a result of the data memory move required by the allgather function. Increasing the clock rate decreases the overhead of the data relocation, but it is still limited by the slower data rate. Conversely, the increased data rate reduces the data transmission time, but it is still limited by the overhead of the slower memory move. These two opposing factors cause the convergence point to be much higher than in any other communication function.

Examination into the relative scalability with respect to system size is achieved by investigation of the average slopes of the curves in Fig. 13. The systems are now examined with a small, single-packet message (i.e. 128B) and a large, multiple-packet message (i.e. 4KB) to identify trends for a range of communication types.

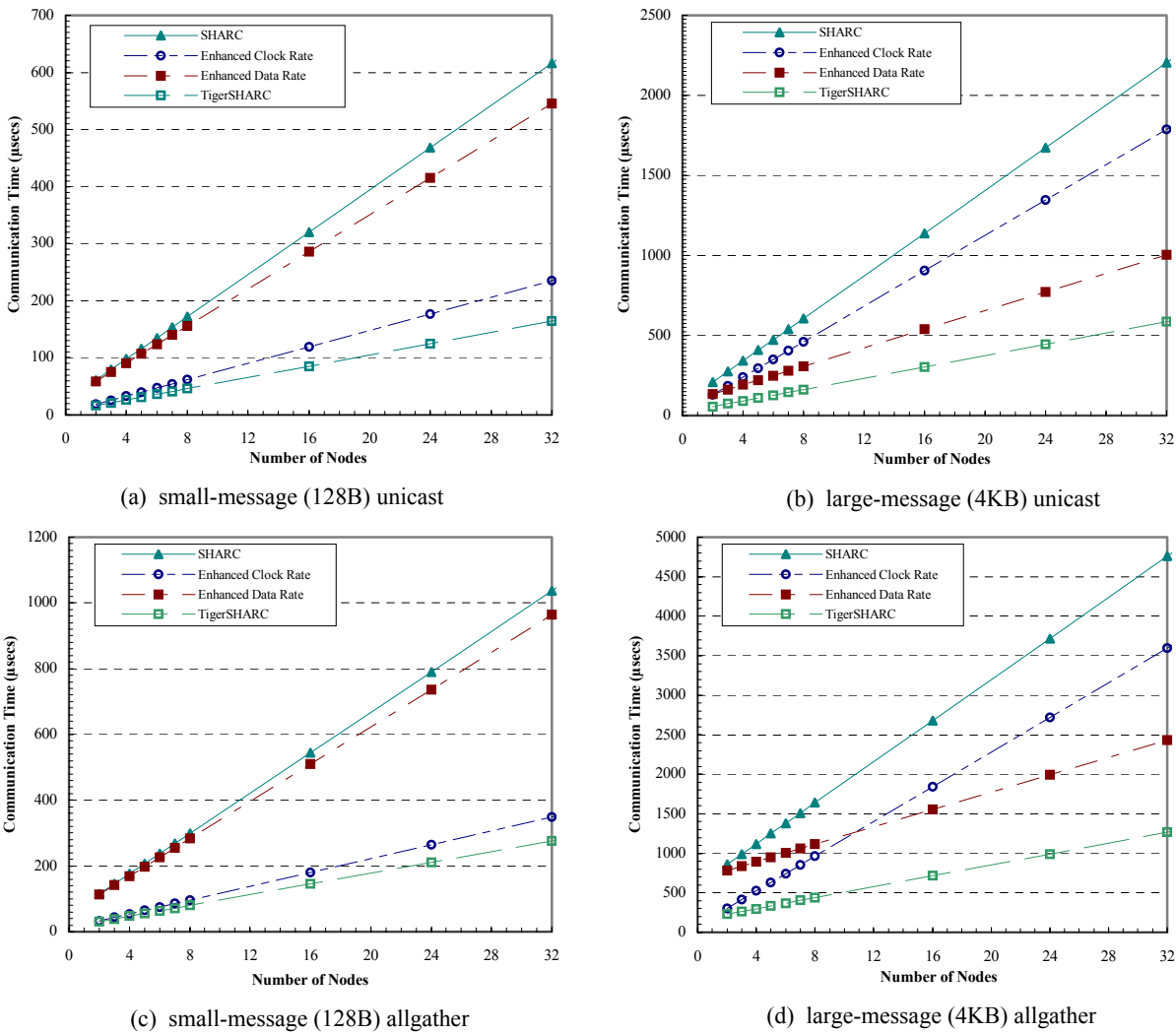


Fig. 13. Scalability of the unicast and allgather communication.

The unicast communications, shown in Figs. 13a and 13b, show the common trend in the scalability analysis. With small messages, the enhanced clock rate system increases at a rate of only 39% that of the original SHARC system, while the enhanced data rate system is much closer to the original increasing at 88%. Conversely, the larger data packets required for the 4KB message causes the enhanced data rate system to increase at a rate of under half the original, while the enhanced clock rate system increases at a rate of 83%.

The small-message allgather communication results, shown in Fig. 13c, demonstrate the same trend with the enhanced clock rate system in achieving a substantial relative decrease in communication time with increasing system size, while the enhanced data rate system stays within 10% of the original. However, the large-message case of Fig. 13d exhibits an interesting crossover phenomenon at an 11-node system size. With smaller message sizes, the magnitude of the *memory move overhead* required for the allgather is substantially reduced as a result of the faster processor clock rate producing a reduced communication time. However, as the number of nodes increases, the amount of data transferred increases while the initial overhead remains constant. The enhanced data rate system reduces the communication time of this additional data and a crossover point is encountered.

A summary of all the scalability measurements is provided in Table 11. Again, 128B and 4KB message sizes are used and the percentages are in comparison to the original SHARC system to determine a relative scalability. The two common trends, where the enhanced clock rate system is beneficial for small message sizes and the enhanced data rate system improves large messages, are apparent in the table. Since the TigerSHARC system improves both the clock rate and data throughput at a consistent rate, the slope of the resulting curves compared to the SHARC is constant. Finally, since the only difference between the in-place allgather and the allgather is a data memory move unrelated to system size, their relative scalability is equal with comparable data sizes.

Table 11
Relative scalability comparison to original SHARC system

Communication Type	Enhanced Clock Rate	Enhanced Data Rate	TigerSHARC
Unicast – small message	39.0%	87.7%	26.7%
Unicast – large message	83.0%	43.7%	26.7%
Broadcast – small message	35.1%	91.6%	26.7%
Broadcast – large message	76.1%	50.6%	26.7%
Allgather – small message	34.3%	92.4%	26.7%
Allgather – large message	84.5%	42.2%	26.7%
In-Place Allgather – small message	34.3%	92.4%	26.7%
In-Place Allgather – large message	84.5%	42.2%	26.7%

7. Conclusion and Future Research

This paper has presented, analyzed, and modeled a lightweight communication service targeted for distributed, embedded DSP systems. By providing a reduced version of the widely accepted MPI protocol, the communication service can be readily employed for systems specifically targeted for an embedded, distributed environment such as sonar beamforming systems. Additionally, by leveraging the architectural features common to DSPs, as well as the

ring topology, the design has proven to produce performance comparable to, and frequently exceeding, MPI services provided on distributed systems based on general-purpose processor machines.

The modeling of the service allowed parameter variation of the system to gain insight without implementing the costly hardware. In the case of the message size study, it was shown that increasing either the network data rate or the processor clock rate of the unicast and allgather functions provided approximately equivalent results. When investigating the relative scalability it was shown that, with small messages, increasing the network data rate by almost a factor of four only decreased the relative scalability of the four functions by an average of 18%. By contrast, increasing the clock rate by the same amount caused the average communication time to increase at a rate of 36% of that of the original. Large message scalability studies exhibited the reverse effect although in a more dispersed pattern. Increasing the processor clock rate reduced the average communication time by only 9%, while an increase data rate produced communication times 45% that of the original.

Future research can proceed in several directions. One area to investigate is the benchmarking of other applications that use more complex message-passing functionality. To further increase performance, especially in the allgather function, the addition of assembly-code optimizations could be implemented. The implementation of an error detection and error correction scheme as well as system fault tolerance should be investigated for a real-world, deployable system. To validate the assumptions made for the TigerSHARC, a simple two- and three-node system could be used to confirm the model parameters for the faster processor. A final area of research would involve a full study to determine the optimum clock and data rates to provide the most efficient power consumption.

Acknowledgements

This work was supported in part by grant N00014-99-1-0278 from the Office of Naval Research, and by equipment and software tools provided by vendor sponsors including Nortel Networks, Intel, Dell, and MPI Software Technology.

References

- [1] A. Alexandrov, M. F. Ionescu, K.E. Schauer, C. Scheiman, LogGP: Incorporating Long Messages into the LogP Model- One step closer towards a realistic model for parallel computation, Proc. 7th ACM Symposium on Parallel Algorithms and Architectures, Santa Barbara, CA, (1995) 95-105.
- [2] Analog Devices, ADSP-TS001 Tiger SHARC DSP Microcomputer Datasheet, Analog Devices, Inc., Norwood, MA, 1998.
- [3] Analog Devices, ADSP-2106x SHARC DSP Microcomputer Family Datasheet, Analog Devices, Inc., Norwood, MA, 1999.
- [4] Bittware Research Systems, User's Guide: Blacktip-EX, Bittware Research Systems, Concord, NH, Revision 0, 1997.
- [5] J. Bruck, D. Dolev, C. Ho, M. Rosu, R. Strong, Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations, Journal of Parallel and Distributed Computing 40 (1) (1997) 19-34.
- [6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauer, E. Santos, R. Subramonian, T. von Eicken, LogP: Towards a Realistic Model of Parallel Computation, Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, CA, (1993) 1-12.
- [7] D. Culler, L.T. Liu, R. Martin, C. Yoshikawa, LogP Performance Assessment of Fast Network Interfaces, IEEE Micro 16 (1) (1996) 35-43.

- [8] A.C. Dusseau, D.E. Culler, K.E. Schauer, R.P. Martin, Fast Parallel Sorting under LogP: Experience with the CM-5, *IEEE Transactions on Parallel and Distributed Systems* 7 (1996) 791-805.
- [9] N. Fang, H. Burkart, MPI-DDL: A distributed-data library for MPI, *Future Generation Computer Systems* 12 (1997) 407-419.
- [10] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, S. Tueke, Wide-area implementation of the Message Passing Interface, *Parallel Computing* 24 (1998) 1735-1749.
- [11] A. George, J. Garcia, K. Kim, P. Sinha, Distributed Parallel Processing Techniques for Adaptive Sonar Beamforming, *Journal of Computational Acoustics*, 10 (1) (2002) 1-23.
- [12] A. George, K. Kim, Parallel Algorithms for Split-Aperture Conventional Beamforming, *Journal of Computational Acoustics*, 7 (4) (1999) 225-244.
- [13] A. George, J. Markwell, Real-time sonar beamforming on high-performance distributed computers, *Parallel Computing* 26 (10) (2000) 1231-1252.
- [14] W. Gropp, Tutorial on MPI: The Message-Passing Interface, Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL.
- [15] W. Gropp, E. Lusk, A high-performance MPI implementation on a shared-memory vector supercomputer, *Parallel Computing* 22 (1997) 1513-1526.
- [16] W. Gropp, E. Lusk, Reproducible Measurements of MPI Performance Characteristics, ANL/MCS-P755-0699, Mathematics and Computer Science Division, Argonne National Laboratory, 1999.
- [17] W. Gropp, E. Lusk, User's guide for MPICH, a portable implementation of MPI, ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.
- [18] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing* 22 (1996) 789-828.
- [19] R. Hempel, D.W. Walker, The emergence of the MPI message passing standard for parallel computing, *Computer Standards & Interfaces* 21 (1999) 51-62.
- [20] H. Bucker, M. Stevenson, Automatic Matched-Field Tracking (AMFT) for Deployable Systems, ONR Peer Review Presentation, MIT Lincoln Labs, Lexington, MA, July 13-15, 1999.
- [21] R. M. Karp, A. Sahay, E. Santos, K.E. Schauer, Optimal Broadcast and Summation in the LogP Model, Technical Report UCB/CSD 92/721, UC Berkeley, 1992.
- [22] K.K. Keaton, T.E. Anderson, D.A. Patterson, LogP Quantified: The Case for Low-Overhead Local Area Networks, *Proc. Hot Interconnects III*, Stanford University, Palo Alto, CA, (1995) 1-16.
- [23] M. Lauria, A.Chien, MPI-FM: High Performance MPI on Workstation Clusters, *Journal of Parallel and Distributed Computing*, 40 (1) (1997) 4-18.
- [24] R. Martin, A.M. Vahdat, D. Culler, T.E. Anderson, Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture, *Proc. 24th Annual International Symposium on Computer Architecture*, Denver, CO, (1997) 85-97.
- [25] T. McMahon, A. Skjellum, eMPI/eMPICH: Embedding MPI, *Proc. Second MPI Developer's Conference*, University of Notre Dame, South Bend, IN, (1996) 57-65.
- [26] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Technical Report CS-94-230, Computer Science Department, University of Tennessee, 1994.
- [27] Message Passing Interface Forum, MPI-2: Extensions to the MPI Interface, Computer Science Department, University of Tennessee, Knoxville, 1997.
- [28] M. Ould-Khaoua, Message latency in the 2-dimensional mesh with wormhole routing, *Microprocessors and Microsystems* 22 (1999) 509-514.
- [29] P. Pacheco, A User's Guide to MPI, Department of Mathematics, University of San Francisco, 1998.
- [30] L. Prylli, The CAPDYN environment and its message-passing library implementation, *Parallel Computing* 23 (1997) 107-120.
- [31] J. Worringer, T. Bemmerl, MPICH for SCI-Connected Clusters, *Proc. SCI-Europe 1999*, Toulouse, France, (1999) 3-11.